Bottom-up Reuse Guidelines: Documentation

Bottom up Reuse Guidelines: Documentation

by James Marshall (Innovim / NASA GSFC)

Based on the presentation, "Bottom-up Reuse: Documentation", by James Marshall to the NASA Earth Science Data Systems Software Reuse Working Group, monthly telecon, December 20, 2006.

Background on Documentation

Documentation provides readers with the additional information they need about the software or system. There are different types of documentation, but the focus here is on user documentation which is for the end users of the software or system. Examples of documentation standards that may apply to user documentation are:

- NASA-STD-2100-91, NASA Software Documentation Standard
- IEEE Std 1063-2001, Standard for User Documentation
- ISO/IEC 18019:2004, Guidelines for the Design and Preparation of User Documentation for Application Software

---

A Reuse Manual

Reusable software components should have a corresponding reuse manual that provides information that enables the evaluation, understanding, use, and adaptation of a software component. A reuse manual answers questions such as:

- What kind of component is it?
- What is the component's functionality?
- Can the component be reused in our context? How?
- What else is needed to reuse the component?
- Can the component be customized/adapted/modified? How? To what extent?
- Can the component be interconnected with our components?
- Is the component's quality sufficient for our purposes?

The reuse manual need not be a real document; it could be an HTML web page, for example. Examples of reuse are always wanted because they offer ways to test software and see how it works, and therefore they should be provided. However, since examples are often the hook that gets people investigating a piece of software, and this job may be performed best when the examples are a separate items.

Internal Documentation and Self-Documenting Code

Documentation my also be internal, included in the source code, rather than external, like the documents described above. Source code is like literature in that it is for humans to read, so readability is a key point to writing good code. Remember that code is not write-once, and you or others will be reading it later. Using a particular coding style/convention can provide a framework for consistent coding, making your code more readable. More readable code is easier to maintain, and this should . Some points that help make code more readable and self-documenting, when used properly, include:

- Good program and logical structure
- Straightforward, easy to understand approaches
- Good choice of names for variables, routines, classes, etc.
- Use of named constants instead of literals
- Clear layout with good formatting
- Minimized control flow and data structure complexity

Benefits of self-documenting code include:

- Simplicity
- Clarity
- Completeness
- Consistency
- Robustness

Self-documenting code should not rely on comments, but comments can be useful additions to code, particularly when they:

- Summarize the code (but not repeat it)
- Describe the code's intent (but not unnecessary explanations)
- Provide information the code can't express itself, such as copyrights, confidentiality notices, version numbers, etc.

Tools such as Javadoc and doxygen can use comments to generate documentation. When appropriate, figures or other visual information should be used to help convey more information compactly.

References

- Software Engineering with Reusable Components by Johannes Sametinger -- types of documentation and reuse manual information
- Code Complete, Second Edition by Steve McConnell -- self-documenting code information
- General Coding Best Practices course on NASA SkillPort (https://nasa.skillport.com/) -- commenting source code, benefits of self-documenting code